AP-377

APPLICATION NOTE

16-Mbit Flash Product Family Software Drivers 28F016SA, 28F016SV, 28F016XS, 28F016XD

TAYLOR GAUTIER
MCD APPLICATIONS ENGINEERING

PATRICK KILLELEA
MCD APPLICATIONS ENGINEERING

SALIM FEDEL
MCD APPLICATIONS ENGINEERING

December 1995

Order Number: 292126-003

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

16-Mbit FLASH PRODUCT FAMILY
SOFTWARE DRIVERS

CONTENTS

PAGE

INTRODUCTION 1

28F016SA C DRIVERS 4

28F016SA ASM86 DRIVERS 26

APPENDIX A: FUNCTION CHANGES A-1

APPENDIX B: GLOSSARY OF TERMS B-1

APPENDIX C: ADDITIONAL INFORMATION C-1

INTRODUCTION

ABOUT THE CODE

This application note provides example software code for word writing, block erasing, and otherwise controlling Intel's 28F016SA, 28F016SV, 28F016XS and 28F016XD (hereafter referred to as 28F016SA) 16 Mbit symmetrically blocked memory components. Two programming languages are provided: high-level "C" for broad platform support, and more optimized ASM86 assembly. In many cases, the driver routines can be inserted "as is" into the main body of code being developed by the system software engineer. Extensive comments are included in each routine to facilitate adapting the code to specific applications.

The internal automation of the 28F016SA makes software timing loops unnecessary and results in platform-independent code. The following example code is designed to be executed in any type of memory and with all processor clock rates. C code can be used with many microprocessors and microcontrollers, while ASM86 assembly code provides a solution optimized for Intel microprocessors and embedded processors.

The 28F016SA, like the 28F008SA, is divided into 64 Kbyte blocks. Since the GSR and BSR are defined relative to the nearest preceding block beginning address, I often refer to this "block base" address in the comments.

Assumptions:

- Pointers (in C) or EDI offsets (in ASM86) are four (4) bytes long, providing a flat addressing space over the entire 28F016SA device. This implies the use of 386 or higher machines. If the code is to be run on a machine with a smaller address space, the code must be modified to include some sort of "windowing" scheme which maps segments of flash into system memory. The Intel 82365 is commonly used for this purpose.
- "Ints" are 16 bit and "longs" 32 bit in C.
- It is assumed that these pointers return a value equal to what they are pointing to. In other words, even though the pointer may be four (4) bytes long, this does not imply that incrementing the pointer by one will move the pointer four (4) bytes in memory. It is entirely dependent upon what the pointer is pointing to that determines how the increment will be effected. In the case of four (4) byte pointers and 16-bit ints, incrementing the pointer by one will effectively move the pointer two (2) bytes.
- There exists a function "set__pin" which can set an individual 28F016SA pin, given the pin number.
- There exists a function "get__pin" which can return the value of an individual 28F016SA pin, given the pin number.
- The C code can access a function which derives the corresponding block base address from any given address.
- BYTE# pin on the device determines whether addressing refers to words or bytes. I assume word writes/reads to a single device. With minor modifications this code can be adapted for a pair of 28F016SAs in Byte mode.
- 28F016SA commands can be written to any address in the block or device to be affected by that command.

Both the C and ASM86 code in this document contain the following routines, in this order:

CSR__word__byte__writes(compatible with 28F008SA)
CSR__block__erase(compatible with 28F008SA)
CSR__erase__suspend__to__read(compatible with 28F008SA)
lock__block
lock__status__upload__to__BSR
update__data__in__a__locked__block
add__data__in__a__locked__block
ESR__word__write
two__byte__write
ESR__page__buffer__write
ESR__block__erase
ESR__erase__all__unlocked__blocks
ESR__suspend__to__read__array
ESR__automatic__erase__suspend__to__write
ESR__full__status__check__for__data__write
ESR__full__status__check__for__erase
single__load__to__pagebuffer
sequential__load__to__pagebuffer
upload__device__information
RYBY__reconfiguration
page__buffer__swap

The names of these routines have been changed to more closely match the algorithms presented in the 28F016SA User's Manual (Order Number 297372). Please see Appendix A for a table documenting these changes.

ABOUT THE 28F016SA

Companion product datasheets for the 28F016SA should be reviewed in conjunction with this application note for a complete understanding of the device.

The example code makes extensive use of bit-masking when interpreting the status registers. As a quick review, note that any bit in a register can be tested by bitwise ANDing the register with the appropriate power of two. Since all of the bits other than the one being

tested are masked out, testing the resulting byte for truth is the same as testing the desired bit for truth. For example, if a register contains 01001010, the test for bit 3 would be ANDing the register with 00001000, or hex 8, and testing the result for truth:

Binary	Hex	
01001010	4A	Register
& 00001000	& 08	Mask for bit 3
= 00001000	= 08	Result

In this case the result byte is true, indicating that bit 3 in the register was a 1.

The meanings of the individual bits of these registers is presented here for reference. Note that there are two status register spaces, both of which are distinct from the flash memory array address space. In the CSR space, the CSR is mapped to every address. In the ESR space, the GSR is mapped two words above the base of each 64K byte block, i.e. to addresses 2, 8002H, 10002H, etc. (in word mode), while each BSR is similarly mapped one word above the base of each 64K byte block to locations 1, 8001H, 10001H, etc. (in word mode), each BSR reflecting the status of its own block.

CSR.7	Write State Machine Status	1 = ready 0 = busy
CSR.6	Erase-suspend Status	1 = erase suspended 0 = erase in progress/ completed
CSR.5	Erase Status	1 = error in block erase 0 = successful block erase
CSR.4	Data-write Status	1 = error in data write 0 = successful data write
CSR.3	V_{PP} Status	1 = V_{PP} low detect/ operation aborted 0 = V_{PP} OK when operation occurred
CSR.2	Reserved for future use	
CSR.1	Reserved for future use	
CSR.0	Reserved for future use	

GSR.7	Write State Machine Status	1 = ready 0 = busy
GSR.6	Operation-suspend Status	1 = operation suspended 0 = operation in progress/ completed
GSR.5	Device Operation Status	1 = operation unsuccessful 0 = operation successful or running
GSR.4	Device Sleep Status	1 = device in sleep 0 = device not in sleep
GSR.3	Queue Status	1 = queue full 0 = queue available
GSR.2	Page Buffer Availability	1 = one/two page buffers available 0 = no page buffers available
GSR.1	Page Buffer Status	1 = selected page buffer ready 0 = selected page buffer busy
GSR.0	Page Buffer Select Status	1 = page buffer 1 selected 0 = page buffer 0 selected

BSR.7	Block Status	1 = ready 0 = busy
BSR.6	Block-lock Status	1 = block unlocked for write/erase 0 = block locked to write/erase
BSR.5	Block Operation Status	1 = error in block operation 0 = successful block operation
BSR.4	Block Operation Abort Status	1 = block operation aborted 0 = block operation not aborted
BSR.3	Queue Status	1 = device queue full 0 = device queue available
BSR.2	V_{PP} Status	1 = V_{PP} low detected 0 = V_{PP} OK when operation occurred
BSR.1	Reserved for future use	
BSR.0	Reserved for future use	

28F016SA Commands

The 28F016SA command set is a superset of the 28F008SA command set, giving existing 28F008SA code the ability to run on the 28F016SA with minimal modifications.

28F008SA-Compatible Commands

00	invalid/reserved
20	single block erase
40	word/byte write
50	clear status registers
70	read CSR
90	read ID codes
B0	erase suspend
D0	confirm/resume
FF	read flash array

28F016SA Performance-Enhancement Commands

0C	page buffer write to flash
71	read GSR and BSRs (i.e. the ESR)
72	page buffer swap
74	single load to page buffer
75	read page buffer
77	lock block
80	abort
96,xx	RY/BY# reconfiguration and SFI configuration (28F016XS)
97	upload BSRs with lock bit
99	upload device information
A7	erase all unlocked blocks
E0	sequential load to page buffer
F0	sleep
FB	two-byte write

28F016XD and 28F016XS Feature Sets

The following features are not supported on the 28F016XD and 28F016XS Fast Flash memories (as compared to the 28F016SA/SV/32SA FlashFile™ memories):

- All page buffer operations (read, load, program, Upload Device Information)
- Command queuing
- Erase All Unlocked Blocks and Two-Byte Write
- Software Sleep and Abort
- RY/BY# reconfiguration via the Device Configuration command

```
"C" DRIVERS
/*************************************************************************/
/* Copyright Intel Corporation, 1993                                     */
/* File : stddefs.h                                                      */
/* Standard definitions for C Drivers for the 28F016SA/SV/XS/XD Flash    */
/* memory components                                                     */
/* Author : Taylor Gautier, Intel Corporation                            */
/* Revision 1.0, 23 September 1994                                       */
/*************************************************************************/


/*************************************************************************/
/* pin values                                                            */
/*************************************************************************/
#define LOW                   0
#define HIGH                  1

/*************************************************************************/
/* error codes                                                           */
/*************************************************************************/
#define NO_ERROR              0
#define VPP_LOW               1
#define OP_ABORTED            2
#define BLOCK_LOCKED          3
#define COMMAND_SEQ_ERROR     4
#define WP_LOW                5

/*************************************************************************/
/* bit masks                                                             */
/*************************************************************************/
#define BIT_0                 0x0001
#define BIT_1                 0x0002
#define BIT_2                 0x0004
#define BIT_3                 0x0008
#define BIT_4                 0x0010
#define BIT_5                 0x0020
#define BIT_6                 0x0040
#define BIT_7                 0x0080

#define LOW_BYTE              0x00FF
#define HIGH_BYTE             0xFF00

/*************************************************************************/
/* RY/BY# enable modes                                                   */
/*************************************************************************/
#define RYBY_ENABLE_TO_LEVEL  1
#define RYBY_PULSE_ON_WRITE   2
#define RYBY_PULSE_ON_ERASE   3
#define RYBY_DISABLE          4

/*************************************************************************/
/* pin numbers                                                           */
/*************************************************************************/
#define WPB 56
/* Write Protect pin (active low) is pin number 56 on standard           */
/* pinout of 28F016SA.                                                   */

#define VPP 15
/* Vpp pin is pin number 15 on standard pinout of 28F016SA.              */
```

292126-1

```
/*************************************************************************/
/* Copyright Intel Corporation, 1993                                     */
/* File : drivers.c                                                      */
/* Example C Routines for 28F016SA/SV/XS/XD Flash memory components       */
/* Original Author : Patrick Killelea, Intel Corporation                 */
/* Revised By : Taylor Gautier, Intel Corporation                        */
/* Revision 2.0, 23 September 1994                                       */
/*                                                                       */
/* NOTE: BYTE# pin on the device determines whether addressing           */
/* refers to words or bytes. I assume word mode.                         */
/* NOTE: A 28F016SA command can be written to any address in the         */
/* block or device to be affected by that command.                       */
/*************************************************************************/
#include <stdio.h>
#include "stddefs.h"

void set_pin(int pin, int level)
{
/* set_pin is an implementation-dependent function which sets a          */
/* given pin on the standard 28F016SA pinout HIGH = 1 or LOW = 0.        */
}

int get_pin(int pin)
{
/* get_pin is an implementation-dependent function which returns a       */
/* given pin on the standard 28F016SA pinout HIGH = 1or LOW  = 0         */
}


int *base(int *address)
{
/* base is an implementation-dependent function which takes an           */
/* address in the flash array and returns a pointer to the base          */
/* of that 64K byte block.                                               */
}

char *byte_base(char *address)
{
/* byte version of base function described above                         */
}
```

292126-2

```
int CSR_word_byte_writes(int *address, int data)
{
/* This procedure writes a byte to the 28F016SA.                    */
/* It also works with the 28F008SA.                                 */
int CSR;
/* CSR variable is used to return contents of CSR register.         */

*address = 0X1010;
/* Word Write command                                               */
*address = data;
/* Actual data write to flash address.                              */
while(!(BIT_7 & *address));
/* Poll CSR until CSR.7 = 1 (WSM ready)                             */

CSR = *address;
/* Save CSR before clearing it.                                     */
*address = 0X5050;
/* Clear Status Registers command                                   */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode. */
return(CSR);
/* Return CSR to be checked for status of operation.                */
}
```

292126-3

```
int CSR_block_erase(int *address)
{
/* This procedure erases a 64K byte block on the 28F016SA.                */
int CSR;
/* CSR variable is used to return contents of CSR register.               */

*address = 0X2020;
/* Single Block Erase command                                             */
*address = 0XD0D0;
/* Confirm command                                                        */
*address = 0xD0D0;
/* Resume command, per latest errata update                               */
while(!(BIT_7 & *address))
/* Poll CSR until CSR.7 = 1 (WSM ready)                                   */
{
/* System may issue an erase suspend command (B0[B0]) here to read data   */
/* from a a different block.                                              */
};

/* At this point, CSR.7 is 1, indicating WSM is not busy.                 */
/* Note that we are still reading from CSR by default.                    */
CSR = *address;
/* Save CSR before clearing it.                                           */
*address = 0X5050;
/* Clear Status Registers command                                         */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.     */
return(CSR);
/* Return CSR to be checked for status of operation.                      */
}
/* If a write has been queued, an automatic erase suspend occurs to write */
/* to a different block.                                                  */
```

292126-4

```
int CSR_erase_suspend_to_read(int *read_address, int *erase_address, int
*result)
{
/* This procedure suspends an erase operation to do a read.                */
int CSR;
/* CSR variable is used to return contents of CSR register.                */

/* Assume erase is underway in block beginning at erase_address.           */
*erase_address = 0XB0B0;
/* Erase Suspend command                                                   */
while(!(BIT_7 & *erase_address));
/* Poll CSR until CSR.7 = 1 (WSM ready)                                    */
if (BIT_6 & *erase_address) {
/* If CSR.6 = 1 (erase incomplete)                                         */
        *erase_address = 0XFFFF;
        /* Read Flash Array command                                        */
        *result = *read_address;
        /* Do the actual read. Any number of reads can be done here.       */
        *erase_address = 0XD0D0;
        /* Erase Resume command                                            */
} else {
        *erase_address = 0XFFFF;
        /* Read Flash Array command                                        */
        *result = *read_address;
        /* Do the actual read. Any number of reads can be done here.       */
}
*erase_address = 0x7070;
/* Read CSR command                                                        */
CSR = *erase_address;
/* Save CSR before clearing it.                                            */
*erase_address = 0X5050;
/* Clear Status Registers command                                          */
return(CSR);
/* Return CSR to be checked for status of operation.                       */
}
```

292126–5

```
int lock_block(int *lock_address)
/* This procedure locks a block on the 28F016SA.                    */
{
int ESR;
/* ESR variable is used to return contents of GSR and BSR.          */

int *block_base = base(lock_address);
/* Find pointer to base of block being locked.                      */

*lock_address = 0X7171;
/* Read Extended Status Registers command                           */
while (BIT_3 & *(block_base + 2));
/* Poll GSR until GSR.3 = 0 (queue available).                      */
set_pin(WPB, HIGH);
/* Disable write protection by setting WPB high.                    */
set_pin(VPP, HIGH);
/* Enable Vpp, wait for ramp if necessary in this system.           */
*lock_address = 0X7777;
/* Lock Block command                                               */
*lock_address = 0XD0D0;
/* Confirmation command                                             */
*lock_address = 0X7171;
/* Read Extended Status Registers command                           */
while (!(BIT_7 & *(block_base + 2)));
/* GSR is 2 words above 0; poll GSR until GSR.7 = 1 (WSM ready).    */

ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.      */
*lock_address = 0X5050;
/* Clear Status Registers command                                   */
*lock_address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode. */
return(ESR);
}
```

292126-6

```
int lock_status_upload_to_BSR(int *address)
/* This procedure uploads status information into the BSR from non-        */
/* volatile status bits.                                                   */
{
int ESR;
/* ESR variable is used to return contents of GSR and BSR.                 */
int *block_base = base(address);
/* Find pointer to base of 32K word block.                                 */

*address = 0X7171;
/* Read Extended Status Registers command                                  */
while (BIT_3 & *(block_base + 2));
/* Poll GSR until GSR.3 = 0 (queue available).                             */
*address = 0X9797;
/* Lock-status Upload command                                              */
*address = 0XD0D0;
/* Confirmation command                                                    */
*address = 0X7171;
/* Read Extended Status Registers command                                  */
while (!(BIT_7 & *(block_base + 2)));
/* Poll GSR until GSR.7 = 1 (WSM not busy)                                 */

ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.             */
*address = 0X5050;
/* Clear Status Registers command                                          */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.      */
return(ESR);
}
```

292126-7

```
int update_data_in_a_locked_block()
{
/* This routine is implemented as pseudo-code to provide an example of     */
/* impementing the flowchart Updating Data in a Locked Block from the      */
/* 28F016SA User's Manual                                                  */

set_pin(WPB, HIGH);
/* set WP# high                                                            */
block_erase_with_CSR(block_address);
/* erase block                                                             */
set_pin(WPB, LOW);
/* set WP# low                                                             */
WriteNewData();
/* Use one of Word/Byte Write, Two-Byte Write or Page Buffer Write to Flash*/

lock_block(block_address);
/* lock block if desired                                                   */
}


int add_data_in_a_locked_block()
{
/* This routine is implemented as pseudo-code to provide an example of     */
/* impementing the flowchart Updating Data in a Locked Block from the      */
/* 28F016SA User's Manual                                                  */
set_pin(WPB, HIGH);
/* set WP# high                                                            */
WriteNewData();
/* Use one of Word/Byte Write, Two-Byte Write or Page Buffer Write to      */
/* Flash                                                                   */
set_pin(WPB, LOW);
/* set WP# low                                                             */
}
```

292126–8

```
int ESR_word_write(int *write_address, int *data, int word_count)
/* This procedure writes a word to the 28F016SA.                          */
{
int counter, ESR;
/* counter is used to loop through data array                             */
/* ESR variable is used to return contents of GSR and BSR.               */
int *block_base = base(write_address);

for (counter = 0; counter < word_count; counter++) {
        *write_address = 0X7171;
        /* Read Extended Status Registers command                         */
        while (BIT_3 & *(block_base + 1));
        /* Poll BSR until BSR.3 of target address = 0 (queue available).  */
        /* BSR is 1 word above base of target block in status reg space.  */
        *write_address = 0X1010;
        /* Write word command                                             */
        *write_address = data[counter];
        /* Write actual data.                                             */
}

*write_address = 0X7171;
/* Read Extended Status Registers command                                 */
while (!(BIT_7 & *(block_base + 1)));
/* Poll BSR until BSR.7 of target address = 1 (block ready).              */
ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.            */
*write_address = 0X5050;
/* Clear Status Registers command                                         */
*write_address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.     */
return(ESR);
}
```

292126-9

```
int two_byte_write(char *address, char *data, int byte_count)
/* This routine is used when BYTE# is low, i.e. the 28F016SA              */
/* is in byte mode, to emulate a word write.                              */
/* Because of this, commands are given as 0x00XY instead of 0xXYXY as in  */
/* the rest of the code presented here.                                   */
/* *data is a byte array containg the low byte, high byte consecutively of */
/* each word.                                                             */
{
int counter, ESR;
/* ESR variable is used to return contents of GSR and BSR.                */
char *block_base = byte_base(address);
/* Find pointer to base of block.                                         */

for (counter = 0; counter < byte_count; counter++) {
        *address = 0X0071;
        /* Read Extended Status Registers command                         */
        while (BIT_3 & *(block_base + 2));
        /* Poll BSR until BSR.3 of target address = 0 (queue available).  */
        *address = 0X00FB;
        /* Two-byte Write command                                         */
        *address = data[counter++];
        /* Load one byte of data register; A0 = 0 loads low byte, A1 high */
        *address = data[counter];
        /* 28F016SA automatically loads alternate byte of data register   */
}

/* Write is initiated. Now we poll for successful completion.             */
*address = 0X0071;
/* Read Extended Status Registers command                                 */
while (!(BIT_7 & *(block_base + 2)));
/* Poll BSR until BSR.7 of target address = 1 (block ready).              */
/* BSR is 1 word above base of target block in status reg space.          */

ESR = (*(block_base + 4) << 8) + (*(block_base + 2) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.            */
*address = 0X0050;
/* Clear Status Registers command                                         */
*address = 0x00FF;
/* Write FFH after last operation to reset device to read array mode.     */
return(ESR);
}
```

292126-10

```
int ESR_pagebuffer_write(int *address, int word_count)
/* This procedure writes from page buffer to flash.                          */
{
/* This routine assumes page buffer is already loaded.                       */
/* Address is where in flash array to begin writing.                         */
/* Low byte of word count word_count must be 127 or fewer, high must be 0.   */
/* High byte of word count exists for future Page Buffer expandability.      */
int ESR;
/* ESR variable is used to return contents of GSR and BSR.                   */
int *block_base = base(address);
/* Find pointer to base of block to be written.                              */

*address = 0X7171;
/* Read Extended Status Registers command                                    */
while (BIT_3 & *(block_base + 1));
/* Poll BSR until BSR.3 of target address = 0 (queue available).             */
*address = 0X0C0C;
/* Page Buffer Write to Flash command                                        */
*address = word_count;
/* high byte is a don't care, write the low byte                             */
*address = 0;
/* write high byte of word_count which must be 0 (reserved for future use)   */
*address = 0X7171;
/* Read Extended Status Registers command                                    */
while (!(BIT_7 & *(block_base + 1)));
/* Poll BSR until BSR.7 of target address =1 (block ready).                  */

ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.               */
*address = 0X5050;
/* Clear Status Registers command                                            */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.        */
return(ESR);
}
```

292126-11

```
int ESR_block_erase(int *erase_address)
/* This procedure erases a block on the 28F016SA.                    */
{
int ESR;
/* ESR variable is used to return contents of GSR and BSR.           */
int *block_base = base(erase_address);
/* Find address of base of block being erased.                       */

*erase_address = 0X7171;
/* Read Extended Status Registers command                            */
while (BIT_3 & *(block_base + 1));
/* Poll BSR until BSR.3 of erase_address = 0 (queue available).      */
/* BSR is 1 word above base of target block in ESR space.            */

*erase_address = 0X2020;
/* Single Block Erase command                                        */
*erase_address = 0XD0D0;
/* Confirm command                                                   */
*erase_address = 0xD0D0;
/* Resume command, per latest errata update                          */
*erase_address = 0X7171;
/* Read Extended Status Registers command                            */
while (!(BIT_7 & *(block_base + 1)));
/* Poll BSR until BSR.7 of target erase_address = 1 (block ready).   */

ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.       */
*erase_address = 0X5050;
/* Clear Status Registers command                                    */
*erase_address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.*/
return(ESR);
}
```

292126-12

```
int ESR_erase_all_unlocked_blocks(int *device_address, long *failure_list)
/* This procedure erases all the unlocked blocks on a 28F016SA.          */
{
int GSR;
/* Return value will contain GSR in both top and bottom byte.           */
/* 32 bit long pointed to by failure_list is used to return map         */
/* of block failures, each bit representing one block's status.         */
/* device_address points to base of chip.                               */
int block;
/* block is used to hold block count for loop through blocks.           */
long power = 1;

*failure_list = 0;
/* Initialize all 32 bits of failure list long to 0.                    */
*device_address = 0X7171;
/* Read Extended Status Registers                                       */
while (BIT_3 & *(device_address + 1));
/* Poll BSR until BSR.3 of target address = 0 (queue available).        */
*device_address = 0XA7A7;
/* Full-chip erase command                                              */
*device_address = 0XD0D0;
/* Confirm command                                                      */
*device_address = 0X7171;
/* Read Extended Status Registers command                               */
while (!(BIT_7 & *(device_address + 2)));
/* Poll GSR until GSR.7 = 1 (WSM ready)                                 */

for (block = 0; block < 0X0020; block++)
{
       /* Go through blocks, looking at each BSR.5 for operation failure */
       /* and setting appropriate bit in long pointed to by failure list. */
       if (BIT_5 & *(device_address + block * 0X8000 + 1))
       /* Multiply block by 32K words to get to the base of each block.  */
              *failure_list += power;
              /* If the block failed, set that bit in the failure list.  */
       power = power << 1;
       /* Increment to next power of two to access next bit.            */
}

GSR = *(device_address + 2);
*device_address = 0X5050;
/* Clear Status Registers command                                       */
*device_address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.   */
return(GSR);
}
```

292126-13

```
int ESR_suspend_to_read_array(int *address,int *result)
/* This procedure suspends an erase on the 28F016SA.                    */
{
/* Address is assumed to point to location to be read.                  */
/* result is used to hold read value until procedure is complete.       */
int ESR;
/* ESR variable is used to return contents of GSR and BSR.              */
int *block_base = base(address);

*address = 0X7171;
/* Read Extended Status Registers command                               */
while (!(BIT_7 & *(block_base + 1)));
/* Poll BSR until BSR.7 of target address = 1 (block ready).            */
/* BSR is 1 word above base of target block in ESR space.               */
*address = 0XB0B0;
/* Operation Suspend command                                            */
*address = 0X7171;
/* Read Extended Status Registers command                               */
while (!(BIT_7 & *(block_base + 2)));
/* Poll GSR until GSR.7 = 1 (WSM ready).                                */
if (BIT_6 & *(block_base + 2)) {
/* GSR.6 = 1 indicates an operation was suspended on this device,       */
        *address = 0XFFFF;
        /* Read Flash Array command                                     */
        *result = *address;
        /* Read the data.                                               */
        *address = 0XD0D0;
        /* Resume the operation.                                        */
} else {
        *address = 0XFFFF;
        /* Read Flash Array command                                     */
        *result = *address;
        /* Read the data.                                               */
}

*address = 0x7171;
/* Read Extended Status Registers command                               */
ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.          */
*address = 0X5050;
/* Clear Status Registers command                                       */
return(ESR);
}
```

292126-14

```
int ESR_automatic_erase_suspend_to_write(int *write_address, int
*erase_address, int data)
/* This procedure writes to one block while another is erasing.          */
{
int ESR;
/* ESR variable is used to return contents of GSR and BSR.               */
int * block_base = base(erase_address);
/* Find pointer to base of block being erased.                           */

*erase_address = 0X7171;
/* Read Extended Status Register command                                 */
while (BIT_3 & *(block_base + 1));
/* Poll BSR until BSR.3 of target address = 0 (queue available).         */
/* BSR is 1 word above base of target block in ESR space.                */
*erase_address = 0X2020;
/* Erase Block command                                                   */
*erase_address = 0XD0D0;
/* Confirm command                                                       */
*erase_address = 0X7171;
/* Read Extended Status Register command                                 */
while (BIT_3 & *(block_base + 1));
/* Poll BSR until BSR.3 of target address = 0 (queue available).         */
/* BSR is 1 word above base of target block in ESR space.                */
*write_address = 0X4040;
/* Word Write command                                                    */
*write_address = data;
/* Write actual data.                                                    */
/* Erase suspends, write takes place, then erase resumes.                */
*erase_address = 0X7171;
/* Read Extended Status Registers command                                */
while (!(BIT_7 & *(block_base + 1)));
/* Poll BSR until BSR.7 of erase address = 1 (block ready).              */
/* BSR is 1 word above base of target block in status reg space.         */

ESR = (*(block_base + 2) << 8) + (*(block_base + 1) & LOW_BYTE);
/* Put GSR in top byte and BSR in bottom byte of return value.           */
*block_base = 0X5050;
/* Clear Status Registers command                                        */
*block_base = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.    */
return(ESR);
}
```

292126-15

```
int ESR_full_status_check_for_data_write(int *device_address)
{
int errorcode;

*device_address = 0x7171;
/* Read Extended Status Resisters command                          */
while (!(BIT_7 & *(device_address + 2))) ;
/* Poll GSR until GSR.7 = 1 (WSM ready)                            */
/* to make sure data is valid                                      */

if (*(device_address + 1) & BIT_2) errorcode = VPP_LOW;
/* BSR.2 = 1 indicates a Vpp Low Detect                            */
else if (*(device_address + 1) & BIT_4) errorcode = OP_ABORTED;
/* BSR.4 = 1 indicates an Operation Abort                          */

else if (get_pin(WPB) == LOW) errorcode = NO_ERROR;
else if (*(device_address + 1) & BIT_6) errorcode = BLOCK_LOCKED;
/* BSR.6 = 1 indicates the Block was locked                        */
else errorcode = NO_ERROR;

while (!(BIT_7 & *(device_address + 2))) ;
/* Poll GSR until GSR.7 = 1 (WSM ready)                            */
/* make sure chip is ready to accept command                       */

*device_address = 0x5050;
/* Clear Status Registers                                          */

return errorcode;
}
```

292126-16

```
int ESR_full_status_check_for_erase(int *device_address)
{
int errorcode = NO_ERROR;

*device_address = 0x7171;
/* Read Extended Status Resisters command                                  */
while (!(BIT_7 & *(device_address + 2)));
/* Poll GSR until GSR.7 = 1 (WSM ready)                                    */
/* make sure command completed                                             */
if (*(device_address + 1) & BIT_2) errorcode = VPP_LOW;
/* BSR.2 = 1 indicates a Vpp Low Detect                                    */
else if (*(device_address + 1) & BIT_4) errorcode = OP_ABORTED;
/* BSR.4 = 1 indicates an Operation Abort                                  */
else if (get_pin(WPB) == LOW && !(*(device_address + 1) & BIT_6))
                errorcode = BLOCK_LOCKED;
/* BSR.6 = 0 indicates the Block was locked                                */
if (errorcode == NO_ERROR) {
        *device_address = 0x7070;
         /* Read Compatible Status Register                                */

        if ((*device_address & BIT_4) && (*device_address & BIT_5))
        /* CSR.4 and CSR.5 == 1 indicate a command sequence error          */
              errorcode = COMMAND_SEQ_ERROR;
}

while (!(BIT_7 & *(device_address + 2))) ;
/* Poll GSR until GSR.7 = 1 (WSM ready)                                    */
/* make sure device is ready                                               */

*device_address = 0x5050;
/* Clear Status Registers command                                          */

return errorcode;
}
```

292126-17

```
void single_load_to_pagebuffer(int *device_address, char *address, int data)
/* This procedure loads a single byte or word to a page buffer.            */
/* device_address points to base of chip.                                  */
{
*device_address = 0X7171;
/* Read Extended Status Registers command                                  */
while (!(BIT_2 & *(device_address + 2)));
/* Poll GSR until GSR.2 = 1 (page buffer available)                        */
*device_address = 0X7474;
/* Single Load to Page Buffer command                                      */
*address = data;
/* Actual write to page buffer                                             */
/* This routine does not affect status registers.                          */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.      */
}
```

292126-18

```
void sequential_load_to_pagebuffer(int *device_address, char *start_address,
int word_count, int* data)
/* This procedure loads multiple words to a page buffer.                    */
/* device_address points to base of chip.                                   */
{
/* Low byte of word_count must be 127 or fewer, high must be 0.             */
/* word_count is zero-based counting, i.e word_count == 0 loads 1 word,     */
/* word_count == 1 loads 2 words etc.                                       */
/* High byte of word_count exists for future Page Buffer expandability.     */
char counter;
/* counter is used to keep track of words written.                          */

*device_address = 0X7171;
/* Read Extended Status Registers command                                   */
while (BIT_2 & *(device_address + 2));
/* Poll GSR until GSR.2 = 0 (page buffer available).                        */
*device_address = 0XE0E0;
/* Sequential Page Buffer Load command                                      */
*start_address = word_count;
*start_address = 0;
/* Automatically loads high byte of count register                          */
for (counter = 0; counter <= word_count; counter++)
       *(start_address + counter) = data[counter];
/* Loop through data, writing to page buffer.                               */
/* This routine does not affect status registers.                           */
*device_address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.       */
}
```

292126-19

```
int upload_device_information(int *address)
/* This procedure uploads the device revision number to the variable DRC.  */
/* This implementation differs in that it does not loop as in the          */
/* algorithm.  This is so the calling routine can have a chance to do      */
/* error checking instead of looping forever waiting for the device        */
/* complete the operation.                                                 */
{
int DRC = 0;
/* DRC variable is used to return device revision status.                  */
int *block_base = base(address);
/* Find pointer to base of 32K word block.                                 */

*address = 0X7171;
/* Read Extended Status Registers command                                  */
while ((BIT_3 & *(block_base + 2)) && (!(BIT_7 & *(block_base + 2))));
/* Poll GSR until GSR.3 = 0 (queue available) and GSR.7 = 1                */
/* (WSM available).                                                        */
*address = 0X9999;
/* Device information Upload command                                       */
*address = 0XD0D0;
/* Confirmation command                                                    */
*address = 0X7171;
/* Read Extended Status Registers command                                  */
while (!(BIT_7 & *(block_base + 2)));
/* Poll GSR until GSR.7 = 1 (WSM not busy)                                 */
if (BIT_5 & *(block_base+2)) return ((*(block_base+2) & HIGH_BYTE) << 8);
/* if GSR.5 = 1 operation was unsuccessful.  Return GSR and 0 in DRC       */
*address = 0X7272;
/* Swap page buffer to bring buffer with status information to top.        */
*address = 0X7575;
/* Read Page Buffer command                                                */
DRC = (*(block_base + 2) & 0XFF00 << 8) + (*(block_base + 3) & LOW_BYTE);
/* Put GSR in top byte of return value.                                    */
/* User should check GSR for operation success                             */
/* Put device revision code in bottom byte of return value.                */
/* Note that device revision code was read from word 3 in page buffer.     */
*address = 0x5050;
/* Clear Status registers command                                          */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.      */
return(DRC);
}
```

292126-20

```
int RYBY_reconfigure(int *address, int mode)
{
/* this procedure changes the RY/BY# configuration mode to the given mode  */
int GSR;
/* the GSR variable is used to return the value of the GSR                 */

*address = 0x7171;
/* Read Extended Registers command                                         */
while (BIT_3 & *(address+2));
/* Poll GSR until GSR.3 = 0 (queue available)                              */
*address = 0x9696;
/* Enable RY/BY# configuration, next command configures RY/BY#            */
switch (mode) {
        case RYBY_ENABLE_TO_LEVEL:
                *address = 0x0101;
                /* Enable RY/BY# to level mode                             */
                break;

        case RYBY_PULSE_ON_WRITE:
                *address = 0x0202;
                /* Enable RY/BY# to pulse on write                         */
                break;

        case RYBY_PULSE_ON_ERASE:
                *address = 0x0303;
                /* Enable RY/BY# to pulse on erase                         */
                break;

        case RYBY_DISABLE:
        default:
                *address = 0x0404;
                /* Enable RY/BY# to disable                                */
                break;
}
*address = 0X7171;
/* Read Extended Status Registers command                                  */
while (!(BIT_7 & *(address + 2))) ;
/* Poll GSR until GSR.7 = 1 (WSM ready)                                    */
GSR = *(address+2) & LOW_BYTE;
/* put GSR into low byte of return value                                   */
*address = 0x5050;
/* Clear Status registers command                                          */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.      */
return (GSR);
}
```

292126-21

```
int page_buffer_swap(int *address)
{
/* This routine attempts to swap the page buffers, returning the value of  */
/* the GSR before the operation in the upper byte and the value of the GSR */
/* after the operation in the lower byte for comparison                     */
/* For operation to be successful, the following must be true :             */
/* (before) GSR.0 = (after) !GSR.0                                          */
/* (after) GSR.5 = 0                                                        */
int GSR;
/* GSR variable is used to return contents of GSR before and after         */
/* operation                                                                */

*address = 0x7171;
/* Read Extended Registers command                                          */
GSR = *(address+2) << 8;
/* Put GSR into upper byte before page buffer swap                          */
*address = 0x7272;
/* write Page Buffer Swap command                                           */
GSR |= (*(address+2) & LOW_BYTE);
/* Put GSR after operation into low byte for comparison                     */
*address = 0x5050;
/* Clear Status registers command                                           */
*address = 0xFFFF;
/* Write FFH after last operation to reset device to read array mode.       */
return (GSR);
}
```

292126-22

```
ASM86 ASSEMBLY LANGUAGE DRIVERS
;=========================================================================
; Copyright Intel Corporation, 1993
; EXAMPLE ASM86 Drivers for the 28F016SA Flash memory component
; Original Author : Patrick Killelea, Intel Corporation
; Revised By : Taylor Gautier
;
; Revision 2.0, September 26, 1994
;
; NOTE:
; The code assumes 32-bit flat model protected mode for simplicity.
; i.e. ES contains 0 and EDI accesses the entire memory space.
;=========================================================================

TEXT    segment byte    public  'CODE'
        assume  cs:TEXT

; Following is the structure by which all parameters are passed.
params STRUCT
        erase_addr              DD      ?       ; base of block or device to
                                                ; erase
        write_addr              DD      ?       ; address to write to
        write_base              DD      ?       ; base address of block written
                                                ; to
        read_addr               DD      ?       ; address to read from
        read_base               DD      ?       ; base address of block read from
        lock_addr               DD      ?       ; base address of block to lock
        data_addr               DD      ?       ; address of data to write
        data                    DW      ?       ; data word to write
        pagebuffer_start_addr   DB      ?       ; start address in page buffer
        word_count              DW      ?       ; number of words for a multiple
                                                ; read/write
params ENDS

;=========================================================================
; Defines
;=========================================================================
;
;=========================================================================
; Error Codes
;=========================================================================
NO_ERROR                DW      0
VPP_LOW                 DW      1
OP_ABORTED              DW      2
BLOCK_LOCKED            DW      3
COMMAND_SEQ_ERROR       DW      4
WP_LOW                  DW      5

;=========================================================================
; Error Codes
;=========================================================================
RYBY_ENABLE_TO_LEVEL    DW      1
RYBY_PULSE_ON_WRITE     DW      2
RYBY_PULSE_ON_ERASE     DW      3
RYBY_DISABLE            DW      4
;=========================================================================
; MACRO set_pin
; This macro pushes parameters needed for the set_pin routine, calls
```

292126-23

```
; set_pin, and then pops those parameters. set_pin is an implementation-
; dependent function which sets a given pin on the standard 28F016SA
; pinout HIGH = 1 or LOW = 0.
;
; Data needed at the beginning of this macro:
; pin:  28F016SA pin number
; level:       level to set pin
; Trashes : CX
;=============================================================================
MACRO   set_pin pin, level
        push    pin                             ; Push pin number
        push    level                           ; Push logic level of pin
        call    near ptr set_pin                ; Call set_pin
        pop     CX                              ; Pop off parameters
        pop     CX
ENDM
;=============================================================================
; MACRO BSROFF
; This macro takes a pointer and increments it by two bytes.  Use this
; macro for obtaining the offset to the BSR from a block base address
;=============================================================================
MACRO   BSROFF  pointer
        add     pointer, 2                      ; BSR is 2 bytes above base
                                                ; address
ENDM
;=============================================================================
; MACRO GSROFF
; This macro takes a pointer and increments it by four bytes.  Use this
; macro for obtaining the offset to the GSR from a block base address
;=============================================================================
MACRO   GSROFF  pointer
        add     pointer, 4                      ; GSR is 4 bytes above base
                                                ; address
ENDM
;=============================================================================
; MACRO GSRBSROFF
; This macro takes a pointer and increments it by two bytes.  Use this
; macro for obtaining the offset to the GSR from the BSR
;=============================================================================
MACRO   GSRBSROFF pointer
        add     pointer, 2                      ; GSR is 2 bytes from BSR
ENDM
;=============================================================================
; MACRO BSRGSROFF
; This macro takes a pointer and subtracts it by two bytes.  Use this
; macro for obtaining the offset to the BSR from the GSR
;=============================================================================
MACRO   BSRGSROFF pointer
        sub     pointer, 2                      ; BSR is 2 bytes below GSR
ENDM
```

292126-24

```
;=========================================================================
; PROCEDURE    CSR_word_byte_writes
; This procedure writes a byte to the 28F016SA. It also works with a the
28F008SA.
; Param fields needed:
;              params.data: data word to be written
;              params.write_addr: offset of 28F016SA address to write
; Output:      BX: CSR, duplicated in both high and low bytes
;=========================================================================
CSR_word_byte_writes   proc    near
       mov     EDI,params.write_addr
       mov     ES:[EDI],1010H                ; Write To Flash command
       mov     ES:[EDI],params.data
       ; Write data to 28F016SA.

WSM_busy1:
       mov     AX,ES:[EDI]                   ; Read CSR
       test    AX,80H                        ; Look at CSR.7.
       jz      short WSM_busy1               ; Loop while CSR.7 = 0.
       ; Poll CSR until CSR.7 = 1, indicatingthat WSM is ready.

       mov     BX,AX                         ; Return CSR in BX.
       mov     ES:[EDI],5050H                ; Clear Status Registers command
       mov     ES:[EDI],FFFFH                ; Reset device to read array mode
       ret                                   ; Return to calling routine.
CSR_word_byte_writes   endp
```

292126-25

```
;=========================================================================
; PROCEDURE     CSR_block_erase
; This procedure erases a 64K byte block on the 28F016SA.
; It also works with a pair of 28F008SAs.
; Param fields needed:
;               erase_address: offset of base of 28F016SA block to erase
; Output        AX: CSR, duplicated in both high and low bytes
;=========================================================================
CSR_block_erase         proc    near
        mov     EDI,params.erase_addr
        mov     ES:[EDI],2020H                  ; Block Erase command
        mov     ES:[EDI],D0D0H                  ; Erase Confirm command
        mov     ES:[EDI],D0D0H                  ; Resume Command per latest
                                                ; errata update
; Note that it is not strictly necessary to write an erase command to
; the base of a block; any address within the block will do.

WSM_busy2:
        mov     AX,ES:[EDI]                     ; Read CSR.
        test    AX,80H                          ; If CSR.7 = 0, test sets ZF.
        ; System may issue an erase suspend command here to read data from a
        ; different block
        jz      short WSM_busy2                 ; Loop while ZF is set.
        ; Poll CSR until CSR.7 = 1, indicating that WSM is ready.

        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI],FFFFH                  ; Reset device to read array mode
        ret                                     ; Return to calling routine
                                                ; CSR is already in AX
CSR_block_erase endp
```

292126-26

```
;=========================================================================
; PROCEDURE    CSR_erase_suspend_to_read
; This procedure suspends an erase operation to do a read.
; It also works with a pair of 28F008SAs.
; It assumes that erase is underway.
; Param fields needed:
;              params.erase_addr: offset of 28F016SA block to erase
;              params.read_addr: offset of 28F016SA address to read
; Output:      BX: CSR, duplicated in both high and low bytes
;              CX: data read from the address in params.read_addr
;=========================================================================
CSR_erase_suspend_to_read proc near
        mov     EDI,params.erase_addr          ; Set up offset of erase address.
        mov     ES:[EDI],B0B0H                 ; Erase Suspend command

WSM_busy3:
        mov     AX,ES:[EDI]                    ; Read CSR from any address.
        test    AX,80H
        jz      short WSM_busy3
        ; Poll CSR until CSR.7 = 1, indicating that WSM is ready.

        test    AX,40H                         ; test CSR.6
        pushf                                  ; save result for action later

        mov     EDI,params.read_addr           ; Set up offset of read address.
        mov     ES:[EDI],FFFFH                 ; Read Flash command
        mov     CX,ES:[EDI]                    ; Do actual read; put result in
                                               ; CX.
        ; Arbitrary number of reads can be done here.

        popf                                   ; get back the result of CSR.6
        jz      short no_resume_command        ; only resume if operation
                                               ; suspended
        mov     ES:[EDI],D0D0H                 ; Erase Resume command

no_resume_command:
        mov     ES:[EDI],7070H                 ; Read CSR command
        mov     BX,ES:[EDI]                    ; Read CSR from any address.
                                               ; Return CSR in BX.
        ret                                    ; Return to calling routine.
CSR_erase_suspend_to_read endp
```

292126-27

```
;=========================================================================
; PROCEDURE     lock_block
; This procedure locks a block on the 28F016SA.
; Param fields needed:
;               params.lock_addr: offset of base of 28F016SA block to lock
; Output:       BX: GSR in high byte and BSR in low byte
;               AX, DX: trash
;=========================================================================
lock_block      proc    near

        mov     EDI,params.lock_addr            ; Set up offset of address.
        mov     ES:[EDI],7171H                  ; Read ESR command
        GSROFF  EDI                             ; Point EDI to GSR

q_unavailable:
        mov     AX,ES:[EDI]
        test    AX,08H
        jnz     short q_unavailable
        ; Poll GSR while GSR.3 = 1, indicating queue unavailable.

        set_pin 56,1                            ; Disable write protection.
        set_pin 15,1                            ; Enable Vpp
        ; Wait for ramp if necessary.
        mov     ES:[EDI],7777H                  ; Lock Block command
        mov     ES:[EDI],D0D0H                  ; Confirmation command
        mov     ES:[EDI],7171H                  ; Read ESR command

WSM_busy4:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy4
        ; Poll GSR while GSR.7 = 0, indicating WSM_busy.

        mov     BH,AH                           ; Store GSR
        ; Look at BSR.6 to see if block successfully locked.
        BSRGSROFF       EDI                     ; Point EDI to BSR from GSR
        mov     AX,ES:[EDI]                     ; Read BSR
        mov     BL,AL                           ; Store BSR
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI],FFFFH                  ; Reset device to read array mode
        ret                                     ; Return to calling routine.
lock_block      endp
```

292126-28

```
;=========================================================================
; PROCEDURE    lock_status_upload_to_BSR
; This procedure uploads status information into the ESR from non-volatile
; status bits.
; Param fields needed:
;              params.lock_addr: offset of 28F016SA device
; Output:      BX: GSR in high byte and BSR in low byte
;              AX, CX, DX: trash
;=========================================================================
lock_status_upload_to_BSR proc        near
        mov     EDI,params.lock_addr
        mov     ES:[EDI],7171H                  ; Read ESR command.
        GSROFF  EDI                             ; Point EDI to GSR

q_unavailable1:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,08H
        jnz     short q_unavailable1
        ; Poll GSR while GSR.3 = 1, indicating queue unavailable.

        mov     ES:[EDI],9797H                  ; Lock-status Upload command
        mov     ES:[EDI],D0D0H                  ; Confirmation command
        mov     ES:[EDI],7171H                  ; Read ESR command

WSM_busy5:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy5
        ; Poll GSR while GSR.7 = 0, indicating WSM_busy

        mov     AX, ES:[EDI]                    ; Read GSR
        mov     BH, AL                          ; Store in high byte of BX
        BSRGSROFF       EDI                     ; Point EDI to BSR from GSR
        mov     AX,ES:[EDI]                     ; Read BSR
        mov     BL, AL                          ; Store BSR
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI],FFFFH                  ; Reset device to read array mode
        ret                                     ; Return to calling routine.
lock_status_upload_to_BSR endp
```

292126-29

```
;=========================================================================
; PROCEDURE     ESR_word_write
; This procedure writes a word to the 28F016SA.
; Param fields needed:
;               params.write_base: offset of base of 28F016SA block to write
;               params.data: data word to write
;               params.write_addr: offset of 28F016SA address to write
; Output:       BX: GSR in high byte and BSR in low byte
;               AX, BX, CX, : trash
;=========================================================================
ESR_word_write proc    near
        push    SI
        mov     EDI,params.write_addr          ; Set up offset of write address.
        mov     ES:[EDI], 7171H                ; Read Extended Status Registers
                                               ; command
        mov     EBX,params.write_base          ; Get base of block to write
        BSROFF  EDI
        mov     CX, params.word_count
        mov     SI, params.data                ; params.data should be a pointer
                                               ; to an array of data to write to
                                               ; the flash array
q_unavailable2:
        mov     AX,ES:[EBX]                    ; Read BSR
        test    AX,08H
        jne     short q_unavailable2
        ;Loop while BSR.3 of target address  = 1, meaning queue full.

        mov     ES:[EDI], 1010H                ; Write Byte command
        movsw                                  ; write data and increment EDI,
                                               ; SI
        loop    q_unavailable2                 ; loop until CX = 0
        mov     EDI,params.write_base          ; Set up offset of block base
                                               ; in case we wrote to the end of
                                               ; the device, in which case EDI
                                               ; will now be 2 bytes past the
                                               ; device
        mov     ES:[EDI],7171H                 ; Read ESR command
        BSROFF  EDI                            ; Point EDI to BSR

block_busy:
        mov     AX,ES:[EDI]                    ; Read BSR
        test    AX,0080H
        jz      short block_busy
        ;Poll BSR while BSR.7 of target address is 0, meaning block busy

        mov     BL,AL                          ; Store BSR in BL
        GSRBSROFF       EDI                    ; Point EDI to GSR from BSR
        mov     BH,ES:[EDI]                    ; Read GSR and store in BH
        mov     ES:[EDI],5050H                 ; Clear Status Registers command
        mov     ES:[EDI], 0FFFFH               ; Reset device to read array mode
        pop     SI
        ret                                    ; Return to calling routine.
ESR_word_write endp
```

292126-30

```
;=========================================================================
; PROCEDURE     two_byte_write
; This routine is used when BYTE# is low, i.e. the 28F016SA
; is in byte mode, to emulate a word write.
; Param fields needed: (assume existence of byte fields data_high and
; data low)
;               params.write_base: offset of base of 28F016SA block to write
;               params.data_high: high data byte to write
;               params.data_low: low data byte to write
;               params.write_addr: offset of 28F016SA address to write
; Output:       BX: GSR in high byte and BSR in low byte
;               AX, CX, DX: trash
;=========================================================================
two_byte_write proc    near

        mov     EDI,params.write_base           ; Set up offset of address.
        mov     ES:[EDI],0071H                  ; Read ESR command
        BSROFF  EDI                             ; Point EDI to BSR
        mov     CX, params.word_count           ; use word_count as byte_count

q_unavailable3:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,08H
        jnz     short q_unavailable3
        ; Loop while BSR.3 of target address is 1, meaning queue full.

        mov     ES:[EDI],00FBH                  ; Two-byte write command
        ; Write low byte of data word
        mov     EDI,params.write_addr           ; Set up offset of address.
        mov     ES:[EDI],params.data_high
        mov     ES:[EDI],params.data_low

; 28F016SA automatically loads alternate byte of data register and
; initiates write. Now we check for successful completion.

        mov     ES:[EDI],7171H                  ; Read ESR command
        mov     EDI,params.write_base
        BSROFF  EDI                             ; Point EDI to BSR

block_busy2:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,80H
        jz      short block_busy2
        ; Poll BSR while BSR.7 of target address is 0, meaning block busy.

        mov     BH,ES:[EDI]                     ; Read BSR
        GSRBSROFF       EDI                     ; Point EDI to GSR from BSR
        mov     BL,ES:[EDI]                     ; Read and store GSR
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        ret                                     ; Return to calling routine.
two_byte_write endp
```

292126-31

```
;=========================================================================
; PROCEDURE     ESR_pagebuffer_write
; This procedure writes from page buffer to flash.
; Param fields needed:
;               params.write_base: offset of base of 28F016SA block to write
;               params.pagebuffer_word_count: number of words to write to flash
;               params.write_addr: offset of 28F016SA address to write
; Output:       BX: GSR in high byte and BSR in low byte of BX
;               AX, CX, DX: trash
;=========================================================================
ESR_pagebuffer_write   proc    near

        push    SI                              ; Save old SI.
        mov     SI,params.word_count            ; Use SI to count words.

; Address is where in 28F016SA flash array to begin write. The lowest
; byte of this must be identical to the start address in the page buffer.
; Low byte of byte_count must be 256 or fewer, high must be 0.
; High byte exists for future Page Buffer expandability.
        mov     EDI,params.write_base           ; Offset of block base address.
        mov     ES:[EDI],7171H                  ; Read ESR command
        BSROFF  EDI

q_unavailable4:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,8
        jne     short q_unavailable4
        ; Loop while BSR.3 of target address is 1, meaning queue full.

        mov     ES:[EDI],0C0CH                  ; Page Buffer Write command
        mov     ES:[EDI],SI                     ; Write count
        ;Only A0 valid here; low or high byte loaded depending on A0.
        mov     ES:[EDI], 0
        ;A0 internally complemented; alternate byte loads; write starts.
        mov     ES:[EDI],7171H                  ; Read ESR command

block_busy3:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,80H
        jz      short block_busy3
        ;Loop while BSR.7 of target address is 0, meaning block busy.

        mov     BL,ES:[EDI]                     ; Read BSR
        GSRBSROFF       EDI                     ; point EDI to GSR from BSR
        mov     BH,ES:[EDI]                     ; Read GSR
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI], 0FFFFH                ; Reset device to read array mode
        pop     SI                              ; Retrieve old SI.
        ret                                     ; Return to calling routine.
ESR_pagebuffer_write   endp
```

292126-32

```
;=========================================================================
; PROCEDURE    ESR_block_erase
; This procedure erases a block on the 28F016SA.
; Param fields needed:
;              params.erase_addr: offset of base of 28F016SA block to erase
; Output:      BX: GSR in high byte and BSR in low byte
;              AX, DX: trash
;=========================================================================
ESR_block_erase        proc    near
       mov     EDI,params.erase_addr          ; Set up offset of address.
       mov     ES:[EDI],7171H                 ; Read ESR command
       BSROFF  EDI                            ; point EDI to BSR

q_unavailable5:
       mov     AX,ES:[EDI]                    ; Read BSR
       test    AX,08H
       jne     short q_unavailable5
       ; Loop while BSR.3 of target address is 1, meaning queue full.

       mov     ES:[EDI],2020H                 ; Block Erase command
       mov     ES:[EDI],D0D0H                 ; Confirm command
       mov     ES:[EDI],D0D0H                 ; Resume command, per latest
                                              ; errata update
       mov     ES:[EDI],7171H                 ; Read ESR command
       ;Note that EDI still points to BSR

block_busy4:
       mov     AX,ES:[EDI]                    ; Read BSR
       test    AX,80H
       jz      short block_busy4
       ;Loop while BSR.7 of target address = 0, i.e. block busy.

       mov     BL,AL                          ; Store BSR in BL.
       GSRBSROFF      EDI                     ; point EDI to GSR from BSR
       mov     BH,ES:[EDI]                    ; Read GSR, store in BH
       mov     ES:[EDI],5050H                 ; Clear Status Registers command
       mov     ES:[EDI], 0FFFFH               ; Reset device to read array mode
       ret                                    ; Return to calling routine.
ESR_block_erase        endp
```

292126-33

```
;=========================================================================
; PROCEDURE     ESR_erase_all_unlocked_blocks
; This procedure erases all the unlocked blocks on a 28F016SA.
;               params.erase_addr: offset of base of device to erase
; Output:       CX: GSR in both high byte and low byte
;               BX: Failure list
;               AX, DX: trash
;=========================================================================
ESR_erase_all_unlocked_blocks proc    near
        push    SI                              ; Save old SI.
        mov     EDI,params.erase_addr           ; erase_addr should be set to
                                                ; the device address
        mov     ES:[EDI],7171H                  ; Read ESR command
        BSROFF  EDI                             ; point EDI to BSR

q_unavailable6:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,08H
        jnz     short q_unavailable6
        ; Poll BSR while BSR.3 of target address is 1, meaning queue full.

        mov     ES:[EDI],A7A7H                  ; Full-chip Erase command
        mov     ES:[EDI],D0D0H                  ; Confirm command
        mov     ES:[EDI],7171H                  ; Read ESR command
        GSRBSROFF EDI                           ; Point EDI to GSR from BSR

WSM_busy6:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy6
        ;loop until GSR.7 indicates WSM is ready

        mov     AX,ES:[EDI]                     ; Read GSR for operation success
        test    AX,20H
        jz      short operation_successful

        ; If GSR.5 = 1, meaning that the operation was unsuccessful,
        ; go through blocks, looking for the ones which didn't erase.
        xor     SI,SI                           ; Clear SI.
        xor     EBX, EBX                        ; Clear EBX for failure list
        mov     EDX, 1                          ; use EDX as mask to set failures
        mov     EDI, params.erase_addr          ; start at the beginning of the
                                                ; device
        mov     CX, 32                          ; 32 blocks in 28F016SA
        BSRGSROFF       EDI                     ; point EDI to BSR from GSR

look_for_bad_erase:
        ; Looking at each BSR.3 for operation success.
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,08H
        jz      short ok_erased

; record number of bad block here
        or      EBX, EDX

ok_erased:
        shl     EDX
```

292126-34

```
        add     EDI, 10000H                     ; Increment EDI to next block
        loop    look_for_bad_erase

operation_successful:
        mov     EDI, params.erase_address       ; reset EDI to device address
        GSROFF  EDI                             ; point EDI to GSR
        mov     BX, ES:[EDI]
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI], 0FFFFH                ; Reset device to read array mode
        pop     SI
        ret                                     ; Return to calling routine.
ESR_erase_all_unlocked_blocks endp
```

292126-35

```
;=========================================================================
; PROCEDURE    ESR_suspend_to_read_array
; This procedure suspends an erase on the 28F016SA.
; Param fields needed:
;              params.erase_addr: offset of base of erasing 28F016SA block
;              params.read_addr: offset of 28F016SA address to read
; Output:      BX: GSR in high byte and BSR in low byte (of erase block)
;              CX: data read from flash
;              AX, DX: trash
;=========================================================================
ESR_suspend_to_read_array proc       near
        mov     EDI,params.erase_addr
        BSROFF  EDI                             ; point EDI to BSR
        mov     ES:[EDI],7171H                  ; Read ESR command

block_busy5:
        mov     AX,ES:[EDI]                     ; Read BSR
        test    AX,80H
        jz      block_busy5
        ; Loop if BSR.7 of target address is 0, meaning block busy.

        mov     ES:[EDI],0B0B0H                 ; Operation Suspend command
        GSRBSROFF EDI                           ; Point EDI to GSR from BSR

WSM_busy7:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy7
        ; Poll GSR until GSR.7 indicates WSM is ready.

        test    AX,40H
        pushf                                   ; store result for later
        mov     EDI,params.read_addr            ; Set up offset of read address.
        mov     ES:[EDI],FFFFH                  ; Write Read Flash Array command
        mov     CX,ES:[EDI]                     ; Read the data

        mov     EDI,params.erase_addr           ; Set up offset of erase address.
        popf
        jz      short nothing_suspended
        ; If GSR.6 indicates an operation was suspended on this device,
        ; then resume the operation.
        mov     ES:[EDI],0D0D0H                 ; Resume command
nothing_suspended:
        mov     BH,AH                           ; Store GSR in BH.
        sub     EDI, 2                          ; Move EDI down to read BSR.
        mov     BL,ES:[EDI]                     ; Read BSR and store in BL
        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        mov     ES:[EDI],0FFFFH                 ; Reset device to read array mode
        ret                                     ; Return to calling routine.
ESR_suspend_to_read_array endp
```

292126-36

```
;=========================================================================
; PROCEDURE    ESR_automatic_erase_suspend_to_write
; This procedure writes to one block while another is erasing.
; Param fields needed:
;              params.data: data word to write to 28F016SA
;              params.erase_addr: offset of 28F016SA address to erase
;              params.write_addr: offset of 28F016SA address to write
; Output:      BX: GSR in high byte and BSR in low byte
;              AX, DX: trash
;=========================================================================
ESR_automatic_erase_suspend_to_write  proc    near
        mov     EDI,params.erase_addr         ; Set up offset of address.
        mov     ES:[EDI],7171H                ; Read ESR command
        BSROFF  EDI                           ; point EDI to BSR

q_unavailable7:
        mov     AX,ES:[EDI]                   ; Read BSR
        test    AX,08H
        jnz     short q_unavailable6
        ; Loop while BSR.3 of target address is 1, meaning queue full.

        mov     ES:[EDI],2020H                ; Write Erase Block command
        mov     ES:[EDI],D0D0H                ; Erase Confirm command
        mov     EDI,params.write_addr         ; Set up offset of address.
        mov     ES:[EDI],4040H                ; Write Word command
        mov     ES:[EDI],params.data          ; Write actual data

; Erase will suspend, write will take place, then erase resumes.

        mov     EDI,params.erase_addr         ; Set up offset of address.
        mov     ES:[EDI],7171H                ; Read ESR command
        BSROFF  EDI                           ; point EDI to BSR

block_busy6:
        mov     AX, ES:[EDI]
        test    AX,80H
        jz      short block_busy6
        ; Loop while BSR.7 of target address is 0, meaning block busy.

        mov     BH,ES:[EDI]                   ; Read BSR
        GSRBSROFF EDI
        mov     BL,ES:[EDI]                   ; Read and store GSR
        mov     ES:[EDI],5050H                ; Clear Status Registers command
        mov     ES:[EDI],0FFFFH               ; Reset device to read array mode
        ret                                   ; Return to calling routine.
ESR_automatic_erase_suspend_to_write  endp
```

292126-37

```
;=========================================================================
; PROCEDURE    ESR_full_status_check_for_data_write
; This procedure performs a full status check of the Extended Status
; Register
; Param fields needed:
;              params.write_base: offset of base of device
; Output:      CX : errorcode
;              AX, BX: trash
;=========================================================================
ESR_full_status_check_for_data_write proc near
       mov     EDI, params.write_base
       mov     ES:[EDI], 7171H                 ; Read ESR command
       GSROFF  EDI                             ; Point EDI to GSR

WSM_busy8:
       mov     AX,ES:[EDI]                     ; Read GSR
       test    AX,80H
       jz      short WSM_busy8
       ; Poll GSR while GSR.7 = 0, indicating WSM busy.

       BSRGSROFF       EDI                     ; Point EDI to BSR from GSR
       mov     AX,ES:[EDI]
       test    AX, 04H                         ; BSR.2 = 1 indicates VPP_LOW
       jz      vpp_high

       mov     CX, VPP_LOW
       jmp     cont

vpp_high:
       test    AX, 10H                         ; BSR.4 = 1 indicates operation
       jz      op_not_aborted                  ; was aborted

       mov     CX, OP_ABORTED
       jmp     cont

op_not_aborted:
       get_pin WPB                             ; get_pin returns output in BX
       cmp     BX, 00H
       je      wpb_high                        ; no error if WPB is low

       jmp     no_error

wpb_high:
       test    AX, 40H                         ; BSR.6 indicates BLOCK_LOCKED
       jz      no_error

       mov     CX, BLOCK_LOCKED
       jmp     cont

no_error:
       mov     CX, NO_ERROR

cont:
       GSRBSROFF       EDI                     ; Point EDI to GSR from BSR

WSM_busy9:
       mov     AX, ES:[EDI]
```

292126-38

```
        test    AX, 80H
        jz      WSM_busy9
        ; Poll GSR while GSR.7 = 0, indicating WSM busy.

        mov     ES:[EDI],5050H                ; Clear Status Registers command
        ret
ESR_full_status_check_for_data_write endp
```

292126-39

```
;=========================================================================
; PROCEDURE    ESR_full_status_check_for_erase
; This procedure performs a full status check of the Extended Status
; Register
; Param fields needed:
;              params.write_base: offset of base of device
; Output:      CX : errorcode
;              AX, BX: trash
;=========================================================================
ESR_full_status_check_for_erase proc near
        mov     CX, NO_ERROR
        mov     EDI, params.write_base
        mov     ES:[EDI], 7171H                 ; Read ESR command
        GSROFF  EDI                             ; Point EDI to GSR

WSM_busy10:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy10
        ; Poll GSR while GSR.7 = 0, indicating WSM busy.

        BSRGSROFF       EDI                     ; Point EDI to BSR from GSR
        mov     AX,ES:[EDI]
        test    AX, 04H                         ; BSR.2 = 1 indicates VPP_LOW
        jz      vpp_high

        mov     CX, VPP_LOW
        jmp     cont

vpp_high:
        test    AX, 10H                         ; BSR.4 = 1 indicates operation
        jz      op_not_aborted                  ; was aborted

        mov     CX, OP_ABORTED
        jmp     cont

op_not_aborted:
        get_pin WPB                             ; get_pin returns output in BX
        cmp     BX, 00H
        jne     no_error                        ; BSR.6 and WPB LOW indicates
        test    AX, 40H                         ; BLOCK_LOCKED
        jz      no_error

        mov     CX, BLOCK_LOCKED
        jmp     cont

no_error:
        mov     ES:[EDI], 7070H                 ; Read CSR command
        mov     AX, ES:[EDI]
        test    AX, 10H                         ; CSR.4 and CSR.5 indicate
        jz      cont                            ; command sequence error
        test    AX, 20H
        jz      cont
        mov     CX, COMMAND_SEQ_ERROR

cont:
        mov     ES:[EDI], 7171H                 ; Read ESR command
```

292126-40

```
        ; EDI still points to GSR

WSM_busy11:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy11
        ; Poll GSR while GSR.7 = 0, indicating WSM busy.

        mov     ES:[EDI],5050H                  ; Clear Status Registers command
        ret
ESR_full_status_check_for_erase endp
```

292126-41

```
;=========================================================================
; PROCEDURE    single_load_to_pagebuffer
; This procedure loads a single byte or word to a page buffer.
; Param fields needed:
;              params.write_base: offset of base of device
;              params.data: data to be written to page buffer
;              params.pagebuffer_start_addr: byte giving pb location to write
; Output:      AX: trash
;=========================================================================
single_pagebuffer_load proc   near
       mov     EDI,params.write_base          ; Set up offset of base address.
       mov     ES:[EDI],7171H                 ; Read ESR command
       GSROFF  EDI                            ; point EDI to GSR

wait_for_pb:
       mov     AX,ES:[EDI]                    ; Read GSR
       test    AX,04H
       jz      short wait_for_pb
       ; Poll GSR until GSR.2 indicates that a page buffer is available

       mov     ES:[EDI],7474H                 ; Single PB Write command
       ; Actual write to page buffer.
       add     EDI,params.pagebuffer_start_addr            ; Set up offset of
                                                           ; address.
       mov     ES:[EDI],params.data
       ; BP+4 is location in pb to write.
       ret                                    ; Return to calling routine.
single_load_to_pagebuffer endp
```

292126-42

```
;=========================================================================
; PROCEDURE     sequential_load_to_pagebuffer
; This procedure loads the page buffer.
; Param fields needed:
;               params.write_addr: offset of origin of device
;               params.data_addr: pointer to data to be written to pg buffer
;               params.pagebuffer_word_count: number of words to write to pg
buffer
;               params.pagebuffer_start_addr: starting pb address of data to
write
; Output:       AX, BX, DX: trash
;=========================================================================
sequential_load_to_pagebuffer proc    near

        sub     SP,2                            ; Set aside room for counter.
        mov     byte ptr [BP-1],0               ; Clear high byte of counter
                                                ; word.
        push    SI                              ; Save old SI.
        mov     SI,word_count                   ;Put # of words to write in SI.
        ; SP+6 must be 128 or fewer, SP+7 must be 0.
        ; High byte exists for future Page Buffer expandability.
        mov     EDI,params.write_addr           ; Set up offset of device
                                                ; address.
        mov     ES:[EDI],7171H                  ; Read ESR command
        ; Commands to control entire 28F016SA do not need to be written to any
        ; particular address.
        GSROFF  EDI                             ; point EDI to GSR

wait_for_pb2:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,4
        jz      short wait_for_pb2
        ; Poll GSR until GSR.2 indicates that a page buffer is available.

        mov     ES:[EDI],E0E0H                  ; Sequential Page Buffer Load
                                                ; cmd.
        ; Loads high or low byte of count register, depending on A0.
        mov     ES:[EDI],SI                     ; Write
        ; Automatically loads alternate byte of count register.
        mov     ES:[EDI],SI                     ; Write

        ; Loop through data, writing to page buffer.
        mov     byte ptr [BP-1],0               ; Load counter.
        jmp     short compare

not_done:
        mov     DX,word ptr [BP-2]              ; Put current val. of counter in
                                                ; DX.
        mov     AL,params.pagebuffer_start_addr ; Get starting address in pb.
        cbw                                     ; Convert it to a word.
        add     AX,DX                           ; Add to get abs. address in pb.
        cwd                                     ; Convert AX to a double word.
        mov     BP+12,DX                        ; Store segment of pb address
                                                ; (0).
        mov     BP+10,AX                        ; Store offset of pb address.
        mov     AX,word ptr [BP-2]              ; Get current value of counter.
        mov     EBX,params.data_addr            ; Get address of where data is.
        add     EBX,AX                          ; Add value of counter to it.
```

292126-43

```
        mov     ES,word ptr [BX]                ; Put data at that address on
                                                ; stack.
        mov     EDI,params.write_base           ; Set up offset of address.
        mov     ES:[EDI],AX                     ; Write
        inc     word ptr [BP-2]                 ; Increment counter.
compare:
        mov     AX,word ptr [BP-2]              ; Get current value of counter.
        cmp     AX,SI                           ; Compare to final value.
        jl      short not_done

; End of loop.
        pop     SI                              ; Retrieve old SI.
        mov     SP,BP                           ; Retrieve old SP.
        ret                                     ; Return to calling routine.
sequential_load_to_pagebuffer endp
```

292126-44

```
;=========================================================================
; PROCEDURE    upload_device_information
; This procedure uploads the device revision code into the page buffer.
; Param fields needed:
;              params.write_base: offset of 28F016SA device
; Output:      DX: Device revision number
;              AX, CX: trash
;=========================================================================
upload_device_information     proc    near
        mov     EDI,params.write_base
        mov     ES:[EDI],7171H                  ; Read ESR command.
        inc     EDI                             ; Move EDI up to GSR.
        inc     EDI

q_unavailable8:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,8
        jne     short q_unavailable8
        ; Poll GSR while GSR.3 = 1, indicating queue unavailable.

WSM_busy12:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        je      short WSM_busy12
        ; Poll GSR while GSR.7 = 0, indicating WSM busy.

        mov     ES:[EDI],9999H                  ; Lock-status Upload command
        mov     ES:[EDI],D0D0H                  ; Confirmation command
        mov     ES:[EDI],7171H                  ; Read ESR command

WSM_busy13:
        mov     AX,ES:[EDI]                     ; Read GSR
        test    AX,80H
        jz      short WSM_busy13
        ; Poll GSR while GSR.7 = 0, indicating WSM_busy

        mov     ES:[EDI],7272H                  ; Swap Page Buffer command
        mov     ES:[EDI],7575H                  ; Read Page Buffer command
        mov     DX,[params.write_base+3]        ; Put revision number in DX
        ; Revision number is 3 words above write_base in page buffer space.
        ; GSR.5 should be checked for operation success before using revision
        ; number.
        ret                                     ; Return to calling routine.
upload_device_information endp
```

292126-45

```
;=========================================================================
; PROCEDURE     RYBY_reconfiguration
; This procedure reconfigures the RY/BY# output mode
; Param fields needed:
;               params.write_base: offset of 28F016SA device
;               params.data: reconfiguration define
; Output:       AX : GSR
;=========================================================================
RYBY_reconfiguration proc     near
        mov     EDI, params.write_base
        mov     ES:[EDI], 7171H             ; Read ESR command
        GSROFF  EDI                         ; Point EDI to GSR

q_unavailable9:
        mov     AX,ES:[EDI]                 ; Read GSR
        test    AX,8                        ; test bit 3
        jne     short q_unavailable9

        mov     ES:[EDI], 9696H             ; RY/BY# reconfiguration code
        cmp     params.data, 01
        jne     switch1

        mov     ES:[EDI], 0101H             ; Enable RY/BY# to level mode
        jmp     break

switch1:
        cmp     params.data, 02
        jne     switch2

        mov     ES:[EDI], 0202H             ; Enable RY/BY# to pulse on write
        jmp     break

switch2:
        cmp     params.data, 03
        jne     default

        mov     ES:[EDI], 0303H             ; Enable RY/BY# to pulse on erase
        jmp     break

default:
        mov     ES:[EDI], 0404H             ; Enable RY/BY# to disable
break:
        mov     ES:[EDI], 7171H             ; Read ESR command

WSM_busy14:
        mov     AX,ES:[EDI]                 ; Read GSR (EDI already points to
        test    AX,80H                      ; GSR)
        jz      short WSM_busy14

        mov     ES:[EDI],5050H              ; Clear Status Registers command
        mov     ES:[EDI],0FFFFH             ; Reset device to read array mode
RYBY_reconfiguration endp
```

292126-46

```
;=========================================================================
; PROCEDURE    page_buffer_swap
; This procedure swaps the page buffers.
; Param fields needed:
;              params.write_base: offset of 28F016SA device
; Output:      AX : GSR before operation in high byte, GSR in low
;                   byte after operation
;=========================================================================
page_buffer_swap proc  near
        mov     EDI, params.write_base
        mov     ES:[EDI], 7171H              ; Read ESR command
        GSROFF  EDI                          ; Point EDI to GSR

        mov     AH,ES:[EDI]                  ; Read GSR and store in AH

        mov     ES:[EDI], 7272H              ; Page Buffer Swap command

        mov     AL,ES:[EDI]                  ; Read GSR and store in AL
        mov     ES:[EDI],5050H               ; Clear Status Registers command
        mov     ES:[EDI],0FFFFH              ; Reset device to read array mode
page_buffer_swap endp

TEXT    ends
```

292126-47

APPENDIX A
FUNCTION CHANGES

OLD FUNCTION NAME	NEW FUNCTION NAME
compatible__block__erase	CSR__block__erase
compatible__suspend__to__read	CSR__erase__suspend__to__read
compatible__byte__write	CSR__word__byte__writes
ESR__block__erase	N/C
ESR__status__check__after__erase*	ESR__full__status__check__for__erase
ESR__status__check__after__write*	ESR__full__status__check__for__data__write
ESR__suspend__to__read	ESR__suspend__to__read__array
ESR__word__write	N/C
erase__all__unlocked__blocks	ESR__erase__all__unlocked__blocks
lock__block	N/C
status__upload	lock__status__upload__to__BSR
pagebuffer__write__to__flash	ESR__pagebuffer__write
sequential__pagebuffer__load	sequential__load__to__pagebuffer
single__pagebuffer__load	single__load__to__pagebuffer
two__byte__write	N/C
write__during__erase	ESR__automatic__erase__suspend__to__write

NOTE:
*code added in Rev 2.0.

APPENDIX B
Glossary of Terms

BSR:	Block Status Register. Each BSR reflects the status of its 64KB block.
CSR:	Compatible Status Register. The CSR reflects the status of the entire device and is identical in format to the Status Register of the 28F008SA.
EDI:	Extended Data Index register on 80386 and higher CPUs.
ESR:	Extended Status Registers. The GSR and BSRs.
GSR:	Global Status Register. The GSR provides additional information about entire device status.
RY/BY#:	Output pin from the 28F016SA indicating status of current operation.
V_{PP}:	Voltage necessary to program the 28F016SA (12V).
WSM:	Write State Machine. On-board processor automating write, erase and other functions.

APPENDIX C

ADDITIONAL INFORMATION

Order Number	Document
290489	28F016SA Datasheet
292124	AP-375 Upgrade Considerations from the 28F008SA to the 28F016SA
297372	16-Mbit Flash Product User's Manual
292127	AP-378 System Optimization Using the Enhanced Features of the 28F016SA
294016	ER-33 Flash Memory Technology ETOX IV
290528	28F016SV Datasheet
292144	AP-393 28F016SV Compatibility with 28F016SA

REVISION HISTORY

Number	Description
001	Original Version
002	Updated Version of C and ASM code, compatible with 28F016SV/XS/XD
003	Added 28F016XS and 28F016XD Feature Set notice